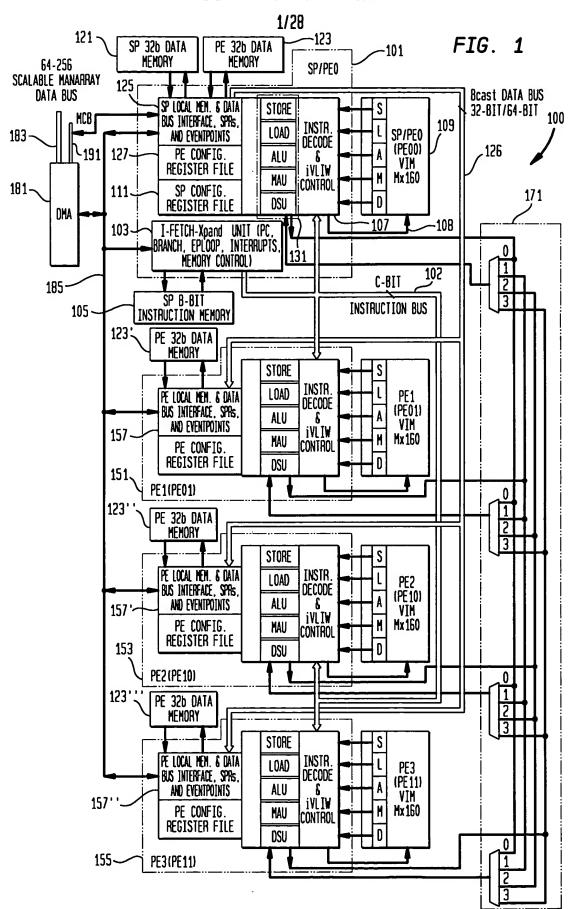
SERIAL NO.: 09/598,566 PETER H. PRIEST (919-806-1600)



SERIAL NO.: 09/598,566 PETER H. PRIEST (919-806-1600)

2/28

FIG. 2A

LSPR ENCODING

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 10 GROUP S/PL/S 110 CE1 SIZE 0 Rt 0 0 0 0 0 0 SPRADDR

FIG. 2B

Syntax/Operation

-)p		
Instruction	Operands	Operation
LOAD ADDRESS REGIST	ER	
LSPR.[SP].W	Rt, SPRADDR	Rt ← [SPRADDR]word
LSPR.[SP].HO	Rt. SPRADDR	Rt.HO ← [SPRADDR]hword
LSPR.[SP].BO	Rt. SPRADDR	Rt.BO ← [SPRADDR]byte
T.LSPR.[SP].[WH080]	Rt. SPRADDR	Do operation only if T condition is satisfied in FO

FIG. 3A

SSPR ENCODING

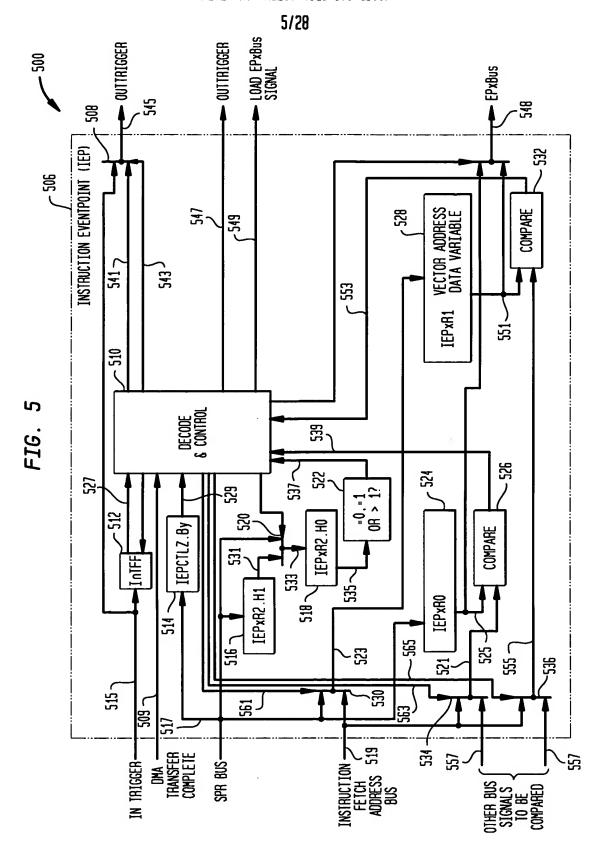
31 30 29 28 27 26	25 24 23 22	21 2	20 19 18 17 16	15 14	13	12 11	10	9876543210
GROUP S/PL/S 110	CE1 SIZE	0 1	Rt	0 0	0	0 0	0	SPRADOR

FIG. 3B

Syntax/Operation

·)		
Instruction	Operands	Operation
STORE SPECIAL-PURPO	SE REGISTER	
SSPR.[SP].W	Rt. SPRADDR	$Rt \rightarrow [SPRADDR]$ word
SSPR.[SP].H0	Rt. SPRADDR	Rt.HO → [SPRADDR]hword
SSPR.[SP].80	Rt. SPRADDR	Rt.BO → [SPRADDR]byte
T.SSPR.[SP].[WH0B0]	Rt. SPRADDR	Do operation only if T condition is satisfied in FO

4/28					
401 7	4037 40	((3111111	EGISTER MAP $ extstyle ag{409}$ $ extstyle $	4
SYSTEM ADDRESS	SP/PE	Sp	PE	DESCRIPTION	400
FOR SP AS A MCB BUS MASTER	LSPR/ SSPR	REGISTER	REGISTER		J
THE NON-SP MCB	ADDRESS				
ADDRESS	7.551.255				
IS THE SAME					
EXCEPT WITH ADDRESS BIT 22=1)					
0x0030080	0x0080	DBSTAT		Debug Status register	
0x0030084	0x0084	DBIR		Debug Instruction register	
0x0030088	0x0088	DBDIN		Debug Data In register	ł.
0x003008c	0x008c	DBDOUT		Debug Data Out register	
0x0030090	0x0090	000007		beday bots out regiote.	
0x0030094	0x0094				
0x0030098	0x0098				
0x003009c	0x009c				1
0x00300a0	0x00a0	EPSTAT	EPSTAT	Event Point Status	1
0x00300a4	0x00a4	DEPCTLO	DEPCTLO	Data Event point Control register 0	1
0x00300a8	0x00a8	DEPCTL1		Data Event point Control register 1	1
0x00300ac	0x00ac	DEPORO	DEPOR0	Data Event point O register O	
0x00300b0	0x00b0	DEPOR1	DEPOR1	Data Event point O register 1	1
0x00300b4	0x00b4	DEPOR2	DEPOR2	Data Event point 0 register 2	
0x00300b8	0x00b8	DEP1R0		Data Event point 1 register 0	1
0x00300bc	0x00bc	DEP1R1		Data Event point 1 register 1	
0x00300c0	0x00c0	DEP1R2		Data Event point 1 register 2	
0x00300c4	0x00c4	DEP2R0		Data Event point 2 register 0	
0x00300c8	0x00c8	DEP2R1		Data Event point 2 register 1	
0x00300cc	0x00cc	DEP2R2		Data Event point 2 register 2	
0x00300d0	0x00d0	IEPCTL0		Instruction Event point Control register 0	
0x00300d4	0x00d4	IEPCTL1		Instruction Event point Control register 1	
0x00300d8	0x00d8	IEP0R0		Instruction Event point 0 register 0	
0x00300dc	0x00dc	IEPOR1		Instruction Event point 0 register 1	} 410
0x00300e0	0x00e0	IEPOR2		Instruction Event point 0 register 2	'''
0x00300e4	0x00e4	IEP1R0		Instruction Event point 1 register 0	!
0x00300e8	0x00e8			Instruction Event point 1 register 1	
0x00300ec	·0x00ec			Instruction Event point 1 register 2	
0x00300f0	0x00f0			Instruction Event point 2 register 0	
0x00300f4	0x00f4			Instruction Event point 2 register 1	!
0x00300f8	0x00f8			Instruction Event point 2 register 2	
0x00300fc	0x00fc			Instruction Event point 3 register 0	
0x0030100	0x0100			Instruction Event point 3 register 1	
0x0030100	0x0104			Instruction Event point 3 register 2	
0x0030100	0x0108			Instruction Event point 4 register 0	
0x0030100	0x010c			Instruction Event point 4 register 1	{
0x0030100	0x0110		l	Instruction Event point 4 register 2	
0x0030100 0x0030100	0x0114	IEP5RO IEP5R1		Instruction Event point 5 register 0	
	0x011B	IEP5R2		Instruction Event point 5 register 1	
0x0030100	0x011c	I TELDUC	L	Instruction Event point 5 register 2	リノ



	602	FIG .	<i>6A</i>	600
	CONTROL VALUE		OPERAT	ION
	00000000	Disabled Event point. No acti OutTrigger ← InTrigger:	on	- 603
604~	-00T11000	OutTrigger ← InTrigger:		//always pass trigger through —— 605
1	This may be used for a loop, with loop skip if count is zero.	if(T==1) InTriggerFF ← InTrigger; ← else InTriggerFF ← 1;	607 608 609	
	If T==1	WHEN((PC==IEPxR0 PC==IEPxR1)	1~	-610
	InTrigger can be used to exit or skip the loop.	if((PC==IEPxR1) &&		//if PC matches "start" address611 //trigger is enabled and active612 //OR 'count' is zero613
) { PC ← IEPxR0; IEPxR2.H0 ← IEPxR2.H1; InTriggerFF ← 0; CancelNext(Inst(PC)); }		//jump to end of loop 614 //reload count 615 //clear FF 616 //Cancel last next fetched inst 617 //(last inst of loop)
	•	else if(PC==IEPxRO)		//if at 'end' address 618
		if(T==1 && InTriggerFF==1)		//if trigger enabled and active ——619
ē		PC ← next PC: IEPxR2.HO ← IEPxR2.H1; InTriggerFF ← O;		//fall out of loop 620 //reload count 621 //clear FF 622
		else if(IEPxR2.H0==0 IEPxR2	!.H0==1) ~	— 623
		PC ← next PC; IEPxR2.H0 ← IEPxR2.H1; ←		624 625
		else		//else Count is neither 1 nor 0 — 626
		PC ← IEPxR1; IEPxR2.H0 ← IEPxR2.H0-1;	•	//next PC is [IEPxR1] 627 //decrement "count" 628
		}		

FIG. 6B

605 7	601 7	640
CONTROL VALUE	OPERAT	TION
00T11001	OutTrigger ← InTrigger;	//always pass trigger through
This is used for a loop which will not skip to the end if the start address is reached and the count is	if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN((PC==IEPxR0 PC==IEPxR1)) {	
zero. If T==1 Trigger can be used to exit or	if({PC==IEPxR1)&&	//if PC matches "start" address //trigger is enabled and active
skip the loop.	PC ← IEPxRO; IEPxR2.H0 ← IEPxR2.H1; InTriggerFF ← 0; CancelNext(Inst[PC]);	//jump to end of loop //reload count //clear FF //Cancel last next fetched inst //(last inst of loop)
	else if(PC==IEPxRO)	//if at "end" address
	if(T==1 && InTriggerFF==1)	//if trigger enabled and active
	PC ← next PC: IEPxR2.H0 ← IEPxR2.H1; InTriggerFF ← 0; } else if(IEPxR2.H0==0 IEPxR2.H0==1) { PC ← next PC: IEPxR2.H0 ← IEPxR2.H1; }	//fall out of loop //reload count //clear FF
	else {	//else Count is neither 1 nor 0
	PC ← IEPxR1; IEPxR2.HO ← IEPxR2.HO-1; } }	//next PC is [IEPxR1] //decrement "count"

FIG. 6C

602 -601 > CONTROL VALUE **OPERATION** 00T11010 OutTrigger ← InTrigger: //always pass trigger through if(T==j) InTriggerFF ← InTrigger: This may be used for a loop else with an exit InTriggerFF ← 1; based on a WHEN((PC==IEPxR0||PC==IEPxR1)) TRUE FO condition or if((PC==IEPxR1) && count == 0, or //if PC matches "start" address.. pre-trigger (if (T==16&InTriggerFF==1) //trigger is enabled and active enabled). If enabled for PC ← IEPxRO; //jump to end of loop pre-trigger, and trigger FF IEPxR2.H0 ← IEPxR2.H1; //reload count is set, and 'start' address InTriggerFF ← 0: //clear FF CancelNext(Inst[PC]): //Cancel last next fetched inst is matched, //(last inst of loop) then the loop is else if(PC==IEPxRO) //if at "end" address skipped. if(T==1 && InTriggerFF==1) //if trigger enabled and active PC ← next PC: //fall out of loop IEPxR2.H0 ← IEPxR2.H1; //reload count //clear FF InTriggerFF ← 0: else if(IEPxR2.H0==0||IEPxR2.H0==1||F0==1 $PC \leftarrow next PC$; if(IEPxR2.H0==0||IEPxR2.H0==1) IEPxR2.H0 ← IEPxR2.H1; } else //else Count is neither 1 nor 0 PC ← IEPxR1; //next PC is [IEPxR1] IEPxR2.H0 ← IEPxR2.H0-1: //decrement "count" }

FIG. 6D

660 602 > 601 > OPERATION CONTROL VALUE 00T11011 OutTrigger ← InTrigger: //always pass trigger through This may be if([==1] used for a loop InTriggerFF ← InTrigger: with an exit else based on a InTriggerFF $\leftarrow 1$: FALSE FO condition or WHEN((PC==IEPxR0 | PC==IEPxR1)) count == 0, or if((PC==IEPxR1) pre-trigger (if //if PC matches "start" address.. enabled). (T==1&&InTriggerFF==1) //trigger is enabled and active If enabled for pre-trigger. and trigger FF PC ← IEPxRO: //jump to end of loop //reload count IEPxR2.H0 ← IEPxR2.H1; is set, and 'start' address //clear FF InTriggerFF ← 0; Cance [Next (Inst[PC]); //Cancel last next fetched inst is matched. then the loop is //(last inst of loop) skipped. else if(PC==IEPxRO) //if at "end" address //if trigger enabled and active if(T==1 && InTriggerFF==1) PC ← next PC: //fall out of loop IEPxR2.H0 ← IEPxR2.H1; //reload count //clear FF InTriggerFF $\leftarrow 0$: else if(IEPxR2.H0==0||IEPxR2.H0==1||F0==0) $PC \leftarrow next PC$; if(IEPxR2.H0==0||IEPxR2.H0==1) IEPxR2.H0 ← IEPxR2.H1; } //else Count is neither 1 nor 0 else PC ← IEPxR1: //next PC is [IEPxR1] IEPxR2.H0 ← IEPxR2.H0-1: //decrement "count"

FIG. 6E

E05 J	7 601	670
CONTROL VALUE		OPERATION
SPT00000	if(P==0)	//Default is pass-through
Used for generating an	OutIrigger ← InTrigger; else OutIrigger ← 0;	//Default is "no trigger"
EP interrupt with optional	if(T==1) InTriggerFF ← InTrigger;	
pre-count, and pre-trigger. Interrupt	else InTriggerFF ← 1;	
occurs just after	WHEN((PC==IEPxRO) (PC==IEP	xR1))&&(!T InTriggerFF))
instruction at address IEPxR1.	if(T && InTriggerFF) InTriggerFF ← 0;	
If SPT=000.	if(IEPxR2.H0 == 0) {	//If match with count=0, carry on //(acts as if disabled)
then this option becomes the	PC ← next PC; } else if(IEPxR2.H0 == 1)	
"No operation" control code	{ if(S==1)	
and no updates to IEP registers	EPINT ← 1; if(P==1)	//assert EP interrupt
occur.	OutTrigger ← 1: IEPxR2,H0 ← IEPxR2.H1;	
	else (
	PC ← next PC: IEPxR2.HO ← IEPxR2.HO - 1	:
)	

FIG. 6F

	. 20.	— 680
605	601 7	000
CONTROL VALUE	OPERA	ATION
0PT00001	if(P==0)	
	OutTrigger← InTrigger;	//Default is pass-through
Used for	else	
vectoring to a	OutTrigger ← 0:	//Default is "no trigger"
target address after 'count'	if(T==1) InTriggerFF ← InTrigger;	
matches	lini iggerii	
ind tories	InTriggerFF ← 1;	
	WHEN((PC==IEPxRO) &&(!T InTriggerFF))
	(
	if(I && InTriggerFF)	//If pre-trigger, then clear FF
	InTriggerFF ← 0;	
	if(IEPxR2.H0 == 0)	//If match with count=0, carry on
	1	//lacts as if disabled)
	PC ← next PC;	
	} -	Hif Watch with samet 4
	else if(IEPxR2.H0 == 1)	//if Match with count 1
	PC ← IEPxR1:	//Branch to vector address
	IEPxR2.H0 ← IEPxR2.H1;	//Reload 'count'
	[) __	
	else	//if neither 0 or 1
	{ PC←next PC:	//next PC
		//decrement 'count'
		rrocorcinent count
	[3]	
	<u> </u>	

FIG. 6G

⁵⁰²	601 7	690
CONTROL VALUE		OPERATION
SPT00010 This IEP can	if(P==0) OutTrigger ← InTrigger; else	//Default is pass-through
be used to generate an EP interrupt after 'count' input trigger events have been received. (If	OutTrigger ← 0; if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN(InTriggerFF))	//Default is 'no trigger'
T==0, then the branch occurs after 'count'	if(T && InTriggerFF) InTriggerFF ← 0:	//If pre-trigger, then clear FF
cycles. repeatedly)	if(IEPxR2.H0 == 0) { PC ← nex t PC; }	<pre>//If match with count=0, carry on //(acts as if disabled)</pre>
	else if(IEPxR2.H0 == 1) { if(S==1)	//if Match with count 1
	EPINT ← 1; if (P==1) Out Trigger ← 1; IEPxR2.H0 ← IEPxR2.H1;	//assert EP interrupt
	else (//If neither 0 or 1
	PC ← nex t PC; IEPxR2.H0 ← IEPxR2.H0 - 1; } }	//next PC //decrement 'count'

SERIAL NO.: 09/598,566 PETER H. PRIEST (919-806-1600)

13/28

FIG. 7A

EPLOOPx
ENCODING

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 10 GROUP S/P Ctrl0p LCF BPID 0 0 0 0 0 0 0 0 0 0 0 0 0 UDISP10

FIG. 7B

Syntax/Operation

Instruction	Operands	Operation
EPL00Px	UDISP10	IEPxR1 ← PC + 1 IEPxR0 ← PC + UDISP10 IEPx ← 0x18 if(IEPxR2.H0>0) { while(IEPxR2.H0>1) { Execute instructions until PC = IEPxR0 PC ← IEPxR1 IEPxR2.H0 ← IEPxR2.H0-1 } Execute instructions until PC = IEPxR0 IEPxR2.H0 ← IEPxR2.H1 } else PC ← PC + UDISP10 + 1

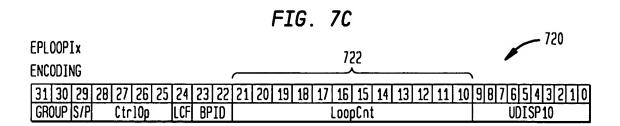


FIG. 7D

- 730

Syntax/Operation	N
------------------	---

Instruction Operands Operation	
IEPxR1 ← PC + 1 IEPxR0 ← PC + UDISP10 IEPxR2.H0 ← LoopCnt IEPxR2.H1 ← loopCnt IEPx ← 0x18 if (IEPxR2.H0>0) { while (IEPxR2.H0>1) { Execute instructions until PC ← IEPxR1 IEPxR2.H0 ← IEPxR2.H0 - 1 } Execute instructions until IEPxR2.H0 ← IEPxR2.H1 } else PC ← PC + UDISP10 + 1	

			FI	6. 8	800
802		804	806	808	810
	ycle	EP Compare	Fetch	Decode	Execute
812 \	0		EPL00Px	Instruction before EPLOOPx	Second instruction before EPLOOPx
814~	1		First instruction of Loop		Instruction before EPLOOPx
				 Calculate EndA=PC+DISP Hold PC and NOP instruction in fetch 	
816 \	2		First instruction of Loop	Hw NOP	EPLOOPx 1. Send EndA to IEPx on SPR bus to be loaded into IEPxRO 2. Signal IEPx to load the program counter value into IEPxR1 3. Hold PC and NOP instruciton in fetch
818 \	3	First compare of IEPxRO to PC here.	First instruction of Loop	NOP	HW NOP
820 ~	4		Next instruction	First instruction of Loop	HW NOP
822 ~	5		:	Next instruction	First instruction of Loop

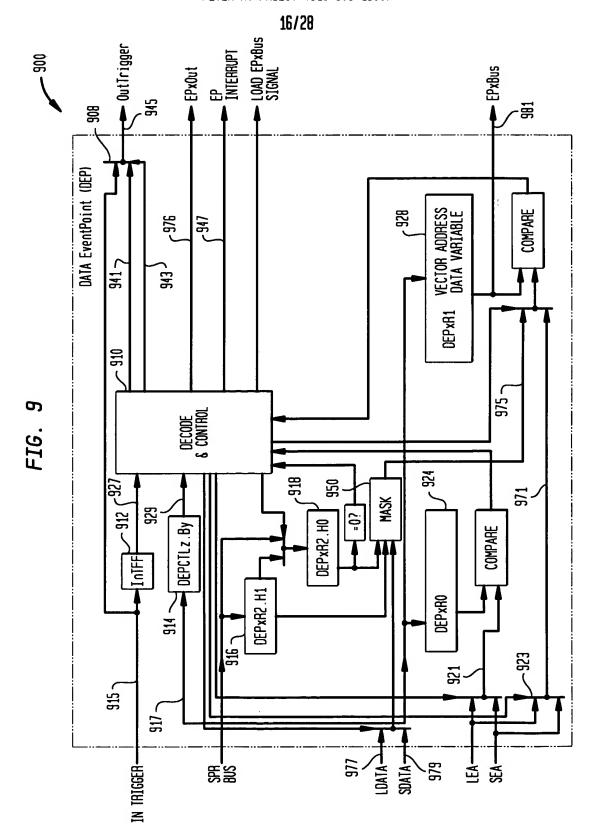


FIG. 10A

- 1010 1012 CONTROL VALUE **OPERATION** if(P==0) SPT01000 OutTrigger ← InTrigger: //Default is pass-through LEA match ATAGLI 33 OutTrigger ← 0: //Default is "no trigger" & MASK) if{{T==1} InTriggerFF ← InTrigger: match after 1 occurance then else interrupt or InTriggerFF ← 1: OutTrigger WHEN((LEA==DEPxRO) &&((LDATA & DEPxR2)==DEPxR1) && InTriggerFF)) if(T && InTriggerFF) InTriggerFF ← 0: //if qualified event, clear FF if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; SPT01001 if(P==0) OutTrigger ← InTrigger: //Default is pass-through SEA match else && ISDATA //Default is "no trigger" OutTrigger ← 0: if([]==1) & MASK) match after 1 InTriggerFF ← InTrigger: 1015 occurance then else interrupt or InTriggerFF ← 1; OutTrigger WHEN((SEA==DEPxRO) &&((SDATA & DEPxR2)==DEPxR1) && InTriggerFF)) if(T && InTriggerFF) InTriggerFF ← 0: //if qualified event, clear FF if (P==1) OutTrigger ← 1; //output 1 cycle pulse

if(S==1) EPINT ← 1;

FIG. 10B

- 1020

1012

CONTROL VALUE OPERATION SPT01010 if(P==0) OutTrigger ← InTrigger: //Default is pass-through (LEA or SEA) OutTrigger ← 0; //Default is "no trigger" match after if(T==1) count InTriggerFF ← InTrigger: occurances Else then interrupt InTriggerFF ← 1; or OutTrigger WHEN(ILEA = DEPxRO) | SEA == DEPxR1) &&(InTriggerFF)) if(I && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0: if (DEPxR2.H0==0) //If match with count=0, carry on //(acts as if disabled) PC ← next PC: else if(DEPxR2.H0==1) //if Match with count 1... if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1: DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' else //If neither 0 or 1... PC ← next PC; DEPxR2.H0 ← DEPxR2.H0 - 1; //decrement 'count'

FIG. 10C

1012 CONTROL VALUE OPERATION SPT01100 if (P==0) OutTrigger ← InTrigger; //Default is pass-through (LEA THEN SEA) match OutTrigger ← 0: //Default is "no trigger" if(T==1) after count occurances InTriggerFF ← InTrigger: then interrupt InTriggerFF ← 1: WHEN(((LEA==DEPxRO)&&(InTriggerFF))AND THEN(SEA == or OutTrigger DEPxR1)) if(I && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0: if(0EPxR2.H0==0) //If match with count 0, carry on //lacts as if disabled) PC ← next PC; else if(DEPxR2.H0==1) //if Match with count 1... if(P==1) OutTrigger ← 1: //output 1 cycle pulse if(S==1) EPINT ← 1; DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' //If neither 0 or 1... else PC ← next PC: DEPxR2.H0 DEPxR2.H0 - 1 //decrement 'count'

- 1030

FIG. 10D

- 1040

10127

CONTROL VALUE **OPERATION** SPT01101 if(P==0) OutTrigger ← InTrigger: //Default is pass-through ISEA THEN //Default is "no trigger" LEA) match OutTrigger ← 0: after count if(T==1) InTriggerFF ← InTrigger: occurances then interrupt else InTriggerFF ← 1: WHEN(((SEA==DEPxR0)&&(InTriggerFF))AND THEN(LEA == DEPxR1)) or OutTrigger if(I && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; //If match with count=0, carry on if(DEPxR2.H0==0) //(acts as if disabled) PC ← next PC; else if(DEPxR2.H0==1) //if Match with count 1... if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1: DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' //If neither 0 or 1... else PC ← next PC: DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count'

)

FIG. 10E

1050

1012

CONTROL VALUE OPERATION SPT01110 if(P==0) OutTrigger ← InTrigger: //Default is pass-through (LEA THEN OutTrigger ← 0; //Default is "no trigger" LEA) match after count if(T==1) InTriggerFF ← InTrigger: occurances then interrupt else InTriggerFF ← 1; WHEN((LEA==DEPxRO)&&(InTriggerFF))AND THEN(LEA == or OutTrigger DEPxR1)) if(I && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(DEPxR2.H0==0) //If match with count 0, carry on //(acts as if disabled) PC ← next PC; else if(DEPxR2.H0==1) //if Match with count 1... if(P==1) OutTrigger ← 1: //output 1 cycle pulse if(S==1) EPINT ← 1: DEPxR2.H1; DEPxR2.H0 //Reload 'count' //If neither 0 or 1... else PC ← next PC: DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count'

FIG. 10F

1060

1012

CONTROL VALUE OPERATION SPT01111 if(P==0) ${\tt OutTrigger} \longleftarrow {\tt InTrigger}; \ {\tt //Default is pass-through}$ ISEA THEN SEA) match //Default is "no trigger" OutTrigger ← 0; if(T==1) after count InTriggerFF ← InTrigger; occurances then interrupt else or OutTrigger InTriggerFF ← 1: WHEN! ((SEA = = DEPxRO) & (InTriggerFF)) AND THEN(SEA == DEPxR1)) if(T && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(DEPxR2.H0==0) //If match with count O, carry on //lacts as if disabled) PC ← next PC; else if(DEPxR2.HO==1) //if Match with count 1... if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINI ← 1: DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' //If neither 0 or 1... else PC ← next PC; DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count' }

FIG. 10G

— 1070

· · · · · · · · · · · · · · · · · · ·			
OPERATION			
if(P==0)			
OutTrigger← InTrigger: //Default is pass-through			
else			
OutTrigger ← 0: //Default is "no trigger"			
if([==1]			
InTriggerFF ← InTrigger;			
else			
InTriggerFF ← 1:			
DEPxRO ← LEA; //save address where data detected			
DETAILS CEA; 77307C BOOK CSS WHELE BOOK BECCERCO			
WHEN((LDATA & DEPxR2) == DEPxR1			
&& InTriggerFF)			
{			
if(T && InTriggerFF)			
InTriggerFF←0; //if qualified event, clear FF			
11.3 14000 000			
Hold(DEPxRO); //Retain state of DEPxRO (LEA)			
if(P==1)			
OutTrigger ← 1: //output 1 cycle pulse			
if (S==1)			
EPINT ← 1:			
}			

FIG. 10H

- 1080

10127

CONTROL VALUE OPERATION SPT10001 if (P==0) OutTrigger ← InTrigger: //Default is pass-through SDATA & //Default is 'no trigger' MASK match OutTrigger ← 0: after 1 if(T==1) InTriggerFF ← InTrigger: occurances capture address else then interrupt InTriggerFF ← 1: and/or OutTrigger. DEPxR0 ← SEA: //save address where data detected Once the WHEN((SDATA & DEPxR2) == DEPxR1 address is held in DEPxRO, it && InTriggerFF) remains held if(T && InTriggerFF) until the InTriggerFF ← 0; //if qualified event, clear FF control value changes or a store to //Retain state of DEPxRO (SEA) Hold (DEPxRO): DEPxRO if(P==1) occurs. OutTrigger ← 1: //output 1 cycle pulse if(S==1) $EPINT \leftarrow 1$;

FIG. 10I

1012

CONTROL VALUE	OPERATION				
SPT 100 10	if(P==0) OutTrigger← InTrigger; //Default is pass-through				
LEA & Mask match.	else OutTrigger←0; //Default is "no trigger" to output				
'T' bit used to specify special actions.	InTriggerFF ← 1; //Always enable event point for this code				
When T==1. count reg treated as a	//Check for DMA access to increment count if(T==1)				
semaphore which can respond to a DMA write address match. A	if((DMA Write Addr & DEPxR1)==(LEA & DEPxR1)//compare after DMA mux DEPxR2.H0 = DEPxR2.H0+1; //increment count				
DMA Write with an address that	//Check for SP/PE access if((DEPxRO & DEPxR1) == (LEA & DEPxR1) && InTriggerFF)				
matches LEA & Mask causes inc of count, while a processor	if(DEPxR2.H0==0) //If match with count=0, carry on { //(acts as if disabled)				
(SP/PE) match event causes dec of count	//do nothing }				
amd an EXTOUT pulse.	else {				
If count==0 when match occurs, no	if(S==1 && P==1) OutTrigger ← 1; //Signal on OutTrigger				
decrement of count occurs and no EXTOUT pulse	if(T==O) //if not using DMA signaling				
(basically no action).	if(DEPxR2.H0 == 1) // check for count reload				
	DEPxR2.H0 ← DEPxR2.H1; // reload count if(S==1 &6 P==0))				
	EPINT ← 1: //if debug int selected, pulse signal }				
	else _DEPxR2.H0 = DEPxR2.H0 -1; //decrement count				
	else //else Using DMA signaling				
	EXTOUT ← 1: Pulse external output 1 cycle DEPxR2.H0 = DEPxR2.H0 -1: //decrement count }				
) }				

FIG. 10J

10127

CONTROL VALUE	ODCOATION
CONTROL VALUE SPT 100 11	if (P==0)
SEA & Mask match. 'T' bit used to specify special actions.	OutTrigger — InTrigger; //Default is pass-through else OutTrigger — O; //Default is "no trigger" to output InTriggerFF — 1; //Always enable event point for this code
When T==1, count reg treated as a semaphore which can respond to a DMA write address match. A DMA Read with an address that matches SEA & Mask causes inc of count, while a processor (SP/PE) match event causes dec of count and an EXTOUT pulse. If count==0 when match occurs, no decrement of count occurs and no EXTOUT pulse (basically no action).	//Check for DMA access to increment count if(T==1) if(OMA Read Addr & DEPxR1) == (SEA & DEPxR1) //compare after DMA mux DEPxR2.H0 = DEPxR2.H0+1;

SERIAL NO.: 09/598,566 PETER H. PRIEST (919-806-1600)

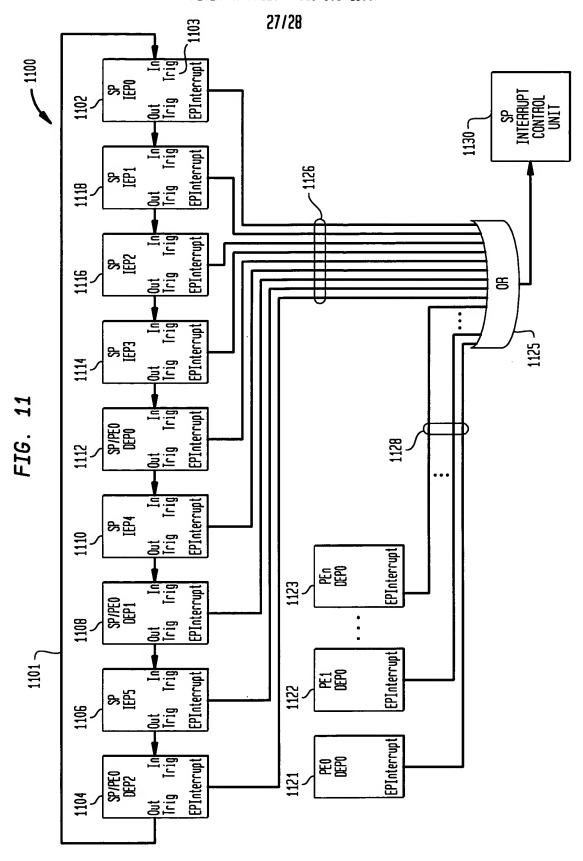


FIG. 12A

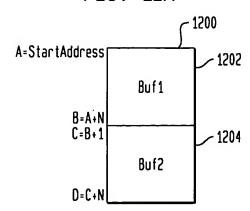


FIG. 12B

1220 -

Initiate DMA for Buf1

LO: Program Routine Start
Processing for the data
in Buf1 or Buf2, use
Load Modulo Index to access
the data from the buffers
LOEND: Has "End of Data: code
been decode? If not branch to
LO, else fall out of LO loop.

FIG. 12C

1240 -

IEPO: DMA not complete event detection
Chained to DEPO and DEP1
IEPORO=notused, IEPOR1=X1, IEPOR2.HO=.H1=2
DEPO: Background DMA to load Buf2
DEPORO=A, DEPOR1=C, DEPOR2.HO=0,.H1=0
DEP1: Background DMA to load Buf1
DEP1RO=C, DEP1R1=A, DEP1R2.HO=1,.H1=0